Statistical Techniques in Robotics (16-831, S21) Lecture #15 (Monday, March 29)

RL & MDP

Lecturer: Kris Kitani *Scribes: Andy Wei, Yi-Chun Chen*

1 Review

In the last lecture, we talked about EXP algorithm in Multi-Armed Bandit problem. In this section, we will give a review of the EXP3[1] and EXP4[1] algorithm.

1.1 EXP3 - Exponential-Weight Update algorithm for Exploration and Exploitation

EXP3 is designed for sampling action in context-free adversarial environment. The algorithm is listed as Algorithm 1 below. It is very similar to Hedge Algorithm, with two difference. First, in exp3 we only know loss for one action in a single round, leading to update only one weight. On the other hand, we are able to acquire loss for all actions in the hedge algorithm and update for all weights. The second difference is that EXP3 directly get reward from taking action, instead of sampling expert.

Algorithm 1 EXP3($\gamma \in [0, 1]$)

$\mathbf{w}^{(1)} \leftarrow \{\mathrm{w}_k^{(1)} = 1\}_{k=1}^{K}$ weights over actions
for $t = 1, \cdots, T$ **do**
  $\mathbf{p}^{(t)} = \frac{\mathbf{w}^{(t)}}{\sum_k w_k^{(t)}}$ probability over actions
  k $\sim$ MULTINOMIAL($\mathbf{p}^{(t)}$) take and draw action
  $a^{(t)} = a_k$
  RECEIVE($r^{(t)} \in [0, 1]$) get reward
  $w_k^{(t+1)} = w_k^{(t)} e^{\gamma \frac{r^{(t)}}{p_k^{(t)}}}$ update weight
end for

1.1.1 Regret Bound

Given Multi-Arm Bandit problem with K possible actions and T round, the regret bound is

$$R \leq O(\sqrt{TK \log K})$$

by setting

$$\gamma = \sqrt{\frac{\log K}{TK}}$$

Thus EXP3 is a no regret algorithm.

1.2 EXP3 Corrected

Another version of EXP3 algorithm is also presented in the lecture. The algorithm is listed as Algorithm 2. The main difference between two version is in line 3, where the corrected version generates probability through a weighted combination between weights and uniform distribution. The regret bound is exactly the same as before and the corresponding optimal γ being

$$\gamma = \sqrt{\frac{T}{2T}}$$

Algorithm 2 EXP3($\gamma \in [0, 1]$)

1: $\mathbf{w}^{(1)} \leftarrow \{\mathrm{w}_k^{(1)} = 1\}_{k=1}^K$ — weights over actions
2: **for** $t = 1, \cdots, T$ **do**
3: $\quad \mathbf{p}^{(t)} = (1-\gamma)\frac{\mathbf{w}^{(t)}}{\sum_k w_k^{(t)}} + \frac{\gamma}{K}$ — probability over actions
4: $\quad \mathrm{k} \sim \mathrm{MULTINOMIAL}(\mathbf{p}^{(t)})$ — take and draw action
5: $\quad a^{(t)} = a_k$
6: $\quad \mathrm{RECEIVE}(r^{(t)} \in [0, 1])$ — get reward
7: $\quad w_k^{(t+1)} = w_k^{(t)} e^{\frac{\gamma}{K}\frac{r^{(t)}}{p_k^{(t)}}}$ — update weight
8: **end for**

1.3 EXP4 - EXP3 with Experts

EXP4 algorithm is designed for contextual Multi-Arm Bandit problem, which can be viewed as a Multi-Arm Bandit problem with N experts. The algorithm for N experts with K actions is listed as Algorithm 3. The regret bound of EXP4 is

$$R \leq \sqrt{KT \log N}$$

which suggests that the algorithm is useful when having few good experts.

Algorithm 3 EXP4($\gamma \in [0, 1], T$)

1: $\mathbf{w}^{(1)} \leftarrow \mathbf{1} \in \mathbb{R}^N$ — weights over **experts**
2: $\mathrm{RECEIVE}(\mathbf{X} \in \mathbb{R}^{N \times K})$ — get K actions advices from N expertss
3: **for** $t = 1, \cdots, T$ **do**
4: $\quad \mathbf{q}^{(t)} = \frac{\mathbf{w}^{(t)}}{\sum_k w_k^{(t)}} \quad \mathbf{X}^{(t)} \in \Delta^K$ — weighted probability of actions over N experts
5: $\quad \mathrm{k} \sim \mathrm{MULTINOMIAL}(\mathbf{q}^{(t)})$ — take and draw action
6: $\quad a^{(t)} = a_k$
7: $\quad \mathrm{RECEIVE}(r^{(t)} \in [0, 1])$ — get reward
8: $\quad \hat{\mathbf{r}}^{(t)} = \frac{r^{(t)}}{q_k^{(t)}} \mathbb{I}[k = k^{(t)}] \in \mathbb{I}^K$ — reward over all arm
9: $\quad \mathbf{g}^{(t)} = \mathbf{X}^{(t)}\ \hat{\mathbf{r}}^{(t)} \in \mathbb{R}^N$ — per expert reward
10: $\quad w_k^{(t+1)} = w_k^{(t)} e^{\gamma g_n^{(t)}} \forall\ n$ — update weight
11: **end for**

2 Summary

2.1 Reinforcement Learning

Problem Formulation

Reinforcement Learning differs from our previous problem with its sequential feedback. The problem can be formulated as

$$\zeta, R \sim D(\zeta, R)$$

where R is the reward and ζ being

$$\zeta_i = \{(x_1, a_1), (x_2, a_2), \ldots, (x_T, a_t)\} R_i \in \mathbb{R}$$

x_i is the current state and a_i is the action to take. The key difference between RL and supervised learning is that every action may affect next state; that being said, all samples in the trajectory are correlated. The goal of our learning algorithm is to learn a mapping from state x to action a

$$\pi : x \to a$$

Review of Learning Problem

Problem	Sampled	Evaluative	Sequential
PWEA	X	X	X
OLC	V	X	X
MAB	X	V	X
C-MAB	V	V	X
RL	V	V	V

The above is the table of learning problems that we have encountered and their attributes.
Comparing to all previous problems, reinforcement learning is sampled, evaluative, and sequential. The problem is sampled as for each time t it sampled the current state ζ from the distribution D. The problem is evaluative as the agent would only get reward of current action, resulting in partially observable loss. The problem is sequential as the current action will affect future state and reward.

2.2 Markov Decision Process

2.2.1 Notations

Before we talk about Markov Decision Process (MDP)[2], let's first look at an concrete example to understand terms state, action, and tracjectory.

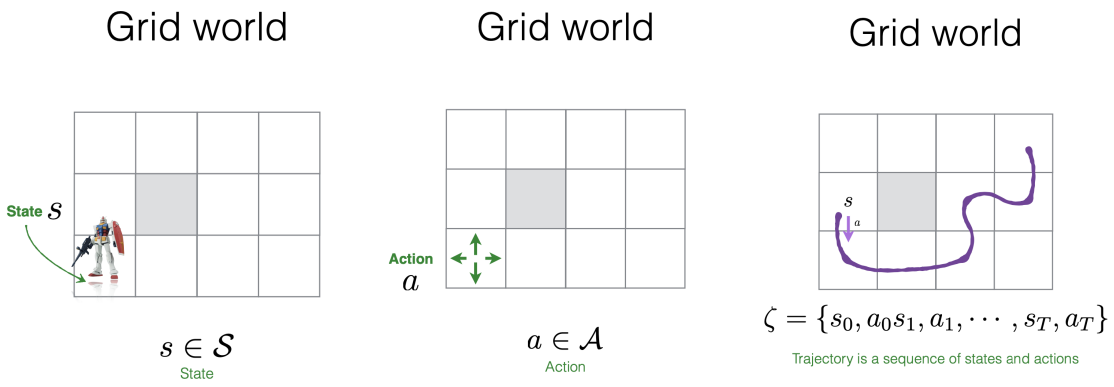

Figure 1: State, action, and trajectory in a grid world.

In this grid world, each cell is a state, and the state is finite. There are actions you can take in each state. In this example, there are 4 actions: East, South, West and North. Trajectory is a sequence of states and actions. Let's also look at some notations needed:

$s \in \mathcal{S}$	*State*
$a \in \mathcal{A}$	*Action*
$p(s^{'}\|s, a)$	*State Transition Dynamic*
$r(s^{'}\|s, a)$	*Reward function*
$p_0(s)$	*State prior*
$\pi(a\|s)$	*Policy*
γ	*Discount factor*

2.2.2 Setup of Markov Decision Process

MDP is way of sequential decision making and we can decompose it to a temporal sequence of variables. Let's define a trajectory as a sequence of states and actions $\zeta = \{s_0, a_0, s_1, a_1, ..., s_T, a_T\}$. Based on the trajectory, we consider the joint distribution of generating a trajectory $p(s_0, a_0, s_1, a_1, ..., s_T, a_T)$ and a reward function $r(s_0, a_0, s_1, a_1, ..., s_T, a_T)$, which is a scalar value for one trajectory. A traditional MDP factorization is as below:

$$p(s_0, a_0, ..., s_T, a_T) = p_0(s_0)\, \Pi_{t=0}^{T-1}\, p(s_{t+1}|s_t, a_t) p(a_t|s_t)$$

The probability distribution over the whole trajectory can be decomposed to the prior state distribution p_0, which is the probability of a state you will start with, and a sequence product of state transition dynamic $p(s_{t+1}|s_t, a_t)$ and policy $p(a_t|s_t)$. We call the reward function over the whole sequence return. There are also different factorization of the return.

$$\begin{aligned} \text{Return} = & r(s_0, a_0, s_1, a_1, ..., s_T, a_T) && \text{or} \\ & r(s_0, a_0, s_1) + r(s_1, a_1, s_2) + ... && \text{or} \\ & r(s_0, a_0) + r(s_1, a_1) + ... && \text{or} \\ & r(s_0) + r(s_1) + ... \end{aligned}$$

, where r(s) is the reward.

We can also look at it with a factor graphical model (Fig. 2).

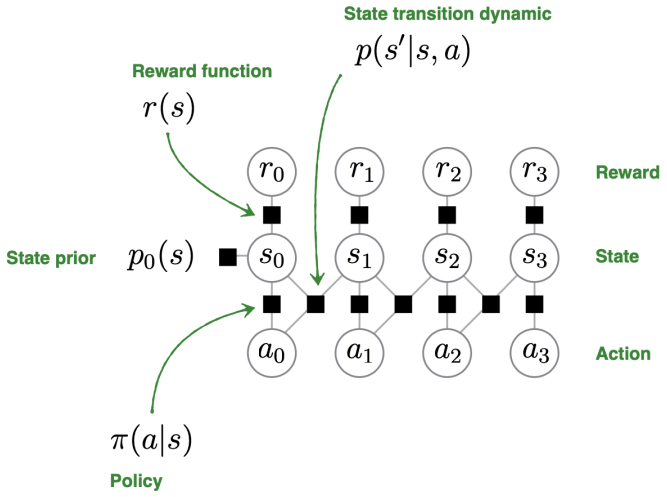

Figure 2: Graphical Model of MDP

Let's also describe the policy, state transition dynamic, and reward in the grid world (Fig. 3). Policy describes which action to take in a given state. State transition dynamic describes the probability of transition to another state. Reward function maps a state to a real value; in this example, we have the reward value finite.

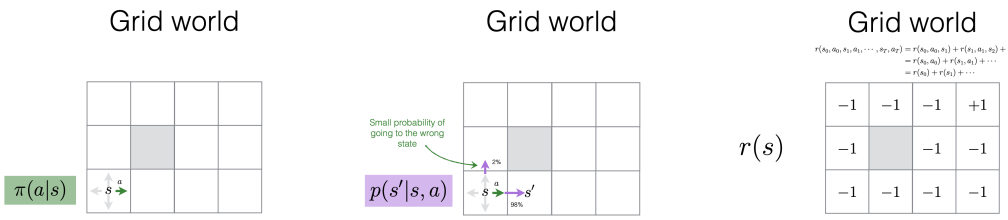

Figure 3: Policy, state transition dynamic, and reward in a grid world.

There are many inference tasks for the MDP. Fig.4 visualize the problems to solve for. Reinforcement learning, where we target at solving for policy, and inverse reinforcement learning, where we target at solving for reward function will be covered in class.

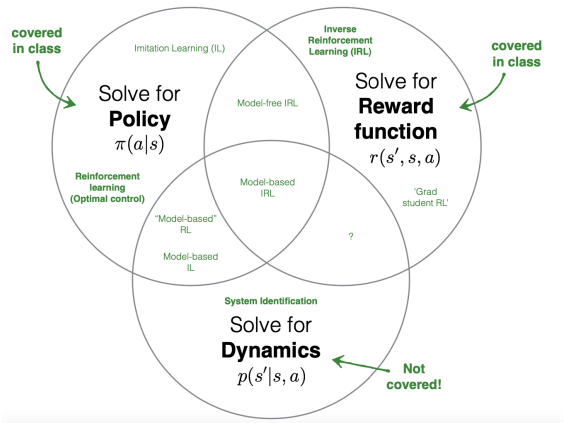

Figure 4: Inference tasks of MDP

There are many RL approaches to find the optimal policy. We can divide it into policy-based, valued-based, and hybrid. There are also many assumptions when finding the optimal policy. Two big categories are:

- Model-based
  - Dynamics known
  - Reward function known
- Model-free
  - Dynamics unknown
  - Reward function unknown
  - Learn through interaction

2.2.3 MDP Concepts

Value Function Value function is to predict the future. A type of value functions is *state value function*, which only depends on states. It is the total expected return of a trajectory in a state s.

$$V^{\pi}(s) = \mathbb{E}_p[r_0 + r_1 + r_2 + ...|s_0 = s]$$

, where $r_t \triangleq r(s_{t+1} = s^{'}, a_t = a, s_t = s)$ and p is the joint distribution of the MDP. Since it i shard to deal with infinite return, we usually have *finite horizon return* and *infinite horizon discounted return*:

$$V^{\pi}(s) = \mathbb{E}_p[r_0 + r_1 + r_2 + ... + r_T|s_0 = s]$$
$$V^{\pi}(s) = \mathbb{E}_p[\gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + ...|s_0 = s]$$

Another type of value function is *state-action value function*. It is the total expected return of a trajectory starting in state s and taking action a.

$$Q^{\pi}(s, a) = \mathbb{E}_p[\gamma^0 r_0(s_0) + \gamma^1 r_1(s_1) + \gamma^2 r_2(s_2) + ...|s_0 = s, a_0 = a]$$

The relationship between V and Q is

$$V^{\pi} = \sum_{a} \pi(a|s) Q^{\pi}(s,a)$$

We use the value function to be the objective function of RL. In other words, we want to find the best policy by maximizing the value function.

$$\begin{aligned} \hat{\pi} &= \arg\max_{\pi} V^{\pi}(s) \\ &= \arg\max_{\pi} \mathbb{E}_{p}[\sum_{t=0} \gamma^{t} r_{t} | s_{0} = s] \end{aligned}$$

References

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[2] R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.